Compliance-Driven Security Automation in AI-Augmented CI/CD Pipelines for Proactive Vulnerability Detection

Harish Apuri, Mukesh Aurangabadkar, Shikher Goel, and Charani Yepuri

[1] Harish Apuri - Software Engineer, Charlotte, North Carolina 2826, USA
hapuri1029@gmail.com
[2] Mukesh Aurangabadkar - Platform Engineer, Denver, Colorado 80134, USA
Mukesh_a24@hotmail.com
[3] Shikher Goel - Lead Software Engineer, Jersey City, New Jersey 07306, USA
Shikher20goel@gmail.com
[4] Charani Yepuri - Independent researcher, Hyderabad, 500001, India
Charaniy04@gmail.com

Abstract. Modern software delivery increasingly adopts CI/CD pipelines that ship changes to production multiple times per day. This high velocity, combined with extensive automation, amplifies security risk because a single insecure commit can propagate to production within minutes. Traditional reactive security testing often runs only nightly or at gate stages and can lag this pace by hours or days and leaving large exposure windows. This paper proposes a quantitative framework for integrating artificial intelligence (AI) including machine learning (ML) and predictive analytics, anomaly detection, and reinforcement learning (RL)—into CI/CD pipelines. This framework enables proactive, continuous, and adaptive vulnerability detection and mitigation and establishing a compliance-driven approach within DevSecOps practices.
Empirical studies show that ML-based code analysis can detect roughly 60% of known vulnerabilities in benchmark datasets and discover classes of issues missed by conventional static analyzers [1]. Furthermore, AI-powered triage can cut false positives by up to half compared with rule-based tools [2] and Similar advances in automated diagnostics demonstrate that AI-driven root-cause analysis systems can reduce mean time to resolution by up to 83.9% compared to manual troubleshooting in cloud-native systems [42]. The paper details an AI-augmented pipeline architecture and outlines metrics such as detection precision/recall, Mean Time-to-Detect (MTTD), and Mean Time-to-Remediate (MTTR), and deployment failure rate to quantify impact. Finally, it identifies future research directions, including hybrid AI-human security review, explainable and compliance-aware models, and adversarially robust AI components for CI/CD security.

Keywords: DevSecOps · CI/CD · Machine Learning · Vulnerability Detection · Reinforcement Learning · Anomaly Detection · Root-Cause Analysis · Compliance Automation

1 Introduction

CI/CD enables software teams to deploy code continuously, with many organizations moving from quarterly releases to daily or hourly deployments per service [3], This rapid pace dramatically compresses the time between introducing a defect and exposing it to users, making traditional security controls—such as SAST, DAST, software composition analysis, at last manual penetration testing inadequate when configured as batch processes [5]. These tools typically run on nightly builds or staging environments, creating systematic blind spots where vulnerabilities introduced between scans can flow through unchecked and generating "security latency" between introduction and detection [1]. DevSecOps addresses this by embedding automated security checks throughout the CI/CD workflow at every stage [6], but conventional rule-based tools struggle with the volume and diversity of modern systems [2]. Artificial intelligence provides mechanisms to address these scalability challenges by learning patterns of secure and insecure behavior directly from data [1, 3], enabling ML models to analyze complex code relationships and assign exploit probabilities [12], build probabilistic models of safe infrastructure configurations [7,8], aggregate signals from multiple sources for container risk scoring [2, 9], and orchestrate multi-agent autonomous workflows to coordinate these security functions end-to-end [43, 47].

Runtime security benefits substantially from AI through anomaly detection algorithms that learn normal operational behavior across multiple layers, flag deviations indicating attacks or misconfigurations [10, 13, 42]. Reinforcement learning can automate response by creating controllers that observe risk scores and choose actions like blocking builds or rolling back deployments and with reward functions balancing security and delivery performance [11,14,15]. AI-driven ranking models prioritize findings by combining contextual features to estimate business impact, ensuring critical controls tied to regulatory standards are enforced deterministically [1,6] and also applying AI introduces challenges including data scarcity, model drift requiring continuous retraining [16, 17], latency constraints, and the need for explainability through techniques like attention visualization and feature importance scoring [18, 19]. AI is not a replacement for traditional controls but a way to make them more scalable, adaptive, and context-aware within DevSecOps pipelines [1, 3, 5].

2 Background and Related Work

Traditional CI/CD security relies on SAST, DAST, software composition analysis, and compliance audits at pre merge or pre release gates [5]. These controls falter in microservice and cloud native environments where deployments occur multiple times daily and forcing teams to trade release velocity for security assurance, Signature-based tools require continuous manual upkeep to track evolving attack techniques and and at scale produce alert noise that drives fatigue and under-triaged findings [2] motivating adaptive and data-driven approaches that generalize beyond static rule sets. ML addresses this gap by classifying vulnerable

code, scoring container images and flagging IaC misconfigurations [1,2,6,44], with studies reporting detection of roughly 60% of known vulnerable lines alongside issues that static analyzers miss [1] and predictive analytics on build telemetry enables proactive gating before high-risk changes reach production [3].

Beyond static analysis, anomaly detection models learn behavioral baselines from CI/CD logs, metrics, and traces to surface deviations such as access spikes or abnormal network flows [10,13]and RL agents increasingly automate response actions including pipeline blocking, release rollbacks and workload isolation [11] though reported reductions in triage effort and MTTD/MTTR vary by deployment context [4]. Complementary advances in AI-driven root-cause analysis demonstrate that fusing logs, metrics, and traces achieves 89.6% diagnostic accuracy and cuts mean time to resolution by 83.9% over manual troubleshooting, leveraging RL feedback loops that improve continuously without requiring large labeled datasets [42]. The core principles underlying these systems multi-modal feature extraction, anomaly clustering and causal propagation translate directly to security diagnostics within CI/CD pipelines and forming the conceptual foundation for the framework proposed in this paper [42].

3 Proposed Framework and Approach

3.1 High-Level Architecture

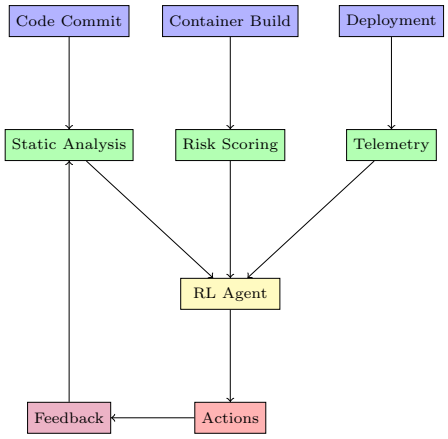

Fig. 1. AI-Augmented DevSecOps Architecture. Data flows from commit to deployment through parallel AI analysis modules, converging at the RL Agent for centralized decision-making and continuous feedback.

The proposed AI-augmented DevSecOps framework distributes AI components across CI/CD stages, starting with a comprehensive data collection and

logging layer [3]. where This layer continuously aggregates source code, pull request metadata, build logs, test outputs, container manifests, IaC definitions, runtime telemetry and aiming to cover the full lifecycle from commit to production. In a typical medium-to-large system where this can mean ingesting thousands of build jobs and millions of log lines per day and requiring scalable storage and streaming analytics infrastructure . The multi-modal data collection approach, informed by recent advances in cloud incident diagnostics, ensures that semantic (logs), numeric (metrics) and structural (traces, topology) signals are all captured [42].

A feature extraction and preprocessing layer transforms these raw artifacts into representations tailored to different models and for example token or graph embeddings for code, dependency graphs for package relationships, and statistical time series for metrics [12]. These features feed specialized modules for static vulnerability detection, container and infrastructure risk scoring and behavioral anomaly detection [1, 2]. Each module outputs quantitative scores (for example, a vulnerability probability or risk score from 0 to 1) that can be thresholded or combined into composite risk indicators per build, deployment, or service.

The final pillar is a self-healing or remediation agent that uses utilizes reinforcement learning (RL) or various types of techniques to map observed risk states to mitigation actions [11,46]. where these actions range from soft measures like adding warnings and creating tickets, to strong interventions such as failing builds, blocking deployments, quarantining containers or triggering automated rollbacks. The framework closes the loop through continuous learning with confirmed vulnerabilities, false positives, near misses, and remediation outcomes are fed back into the models, enabling periodic retraining that adapts to new code patterns and threat tactics over weeks and months [17]. This feedback-driven approach mirrors the reinforcement learning mechanisms employed in advanced cloud RCA systems [42].

3.2 Workflow

When a developer pushes code or opens a merge request, the CI system triggers the AI enhanced pipeline and ML based static analysis runs on the code diff and associated IaC changes, producing vulnerability scores and highlighting specific files or lines, which can be fed into code review tools as annotated findings [1, 45]. The historical studies indicate that integrating such checks at commit or merge time can prevent a significant fraction of vulnerabilities from ever reaching production, reducing downstream incident handling costs [4].

As container images are built, predictive risk models analyze base images, installed packages, configuration flags, and known vulnerability databases to compute a container risk score [2]. The pre release and staging environments are monitored using anomaly detection models that track performance and security related metrics and with unusual spikes in error rates, authentication failures or network traffic patterns can raise early alerts before full traffic exposure [10]. Where in production environment, continuous runtime monitoring and AI driven

SIEM capabilities maintain a feedback channel so that new attack behaviors quickly influence future risk scoring and gating decisions [13].

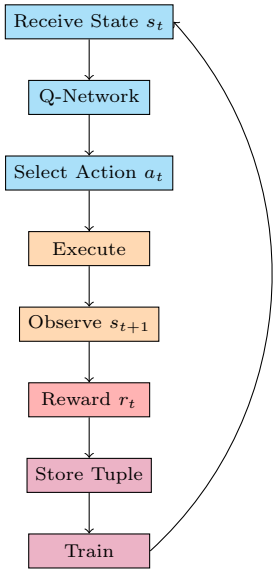

Fig. 2. Reinforcement Learning Agent Decision Cycle. The agent observes pipeline states, selects actions via Q-Network, executes them, calculates rewards based on security and operational metrics, stores experience tuples, and continuously retrains its policy.

When the system detects a vulnerability or anomaly above predefined risk thresholds and the self healing agent evaluates candidate remediation actions according to policies that balance security impact including user experience, and business criticality [11]. The lower risk issues might automatically create tickets and apply non disruptive mitigations and whereas high risk findings in critical services could trigger canary rollbacks or full deployment blocks, subject to human approval where required. All decisions and their downstream outcomes are logged as training signals and enabling the agent to refine its policies to minimize both residual risk and unnecessary interruptions over time [14].

4 Threat Modeling and Compliance Mapping

4.1 STRIDE-Based Threat Analysis

To systematically identify security risks within AI-augmented CI/CD pipelines, we apply the STRIDE threat modeling framework , categorizing threats across six dimensions: Spoofing, Repudiation, Tampering,Information Disclosure, Denial of Service and Elevation of Privilege. All the stages in the pipeline, including

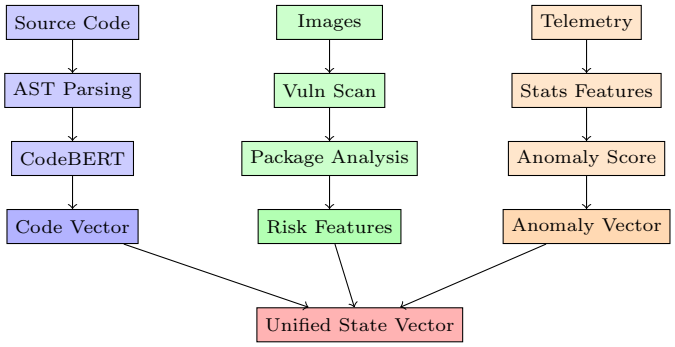

Fig. 3. Feature Engineering and Multi-Modal Data Fusion. Three parallel data processing tracks (code, container, telemetry) converge into a unified state vector for RL agent consumption.

commit, build, test, deploy, and runtime, are analyzed in the context of applicable STRIDE threats. Artificial intelligence components are used as mitigating factors. For instance, the tampering threats in the code commit stage are mitigated through the use of a static analyzer based on machine learning, which detects abnormal code patterns deviating from the established historical baseline. [12].

Table 1. STRIDE Threat Mapping to AI Mitigation Controls

Stage	Threat (STRIDE)	AI Control	Standard
Commit	Tampering	ML Code Classifier	NIST CSF
Build	Elevation	Container Risk Scorer	CIS Docker
Deploy	Spoofing	Anomaly Detection	SOC 2
Runtime	DoS	RL Remediation Agent	ISO 27001
IaC Config	Info Disclosure	IaC Risk Scorer	PCI-DSS

4.2 Regulatory Compliance Alignment

The proposed framework aligns with , ISO/IEC 27001, NIST SP 800-53, SOC 2 Type II and PCI-DSS v4.0 by enforcing compliance controls deterministically through rule-based gates that operate independently of AI models by ensuring critical regulatory requirements are satisfied regardless of model confidence scores. The reinforcement learning agent's reward function incorporates a compliance component (R_{comp}) that penalizes policy violations by -20 points [6] and automated compliance reporting links every security event to the appropriate control identifiers, generating a continuous audit evidence trail and providing compliance officers with real-time compliance dashboards [5].

5 Methodology

5.1 Research Design and Experimental Framework

This research adopts a mixed-methods experimental design to evaluate the proposed AI-augmented DevSecOps framework against conventional rule-based approaches [20]. The main methodology is based on the controlled parallel deployment process, conducted over a period of 12 months through the production software delivery infrastructure, consisting of three distinct phases for the establishment of the baseline (months 1-3), the parallel operation (months 4-6), and the rigorous evaluation (months 7-12). **Research Hypothesis (H1)**: AI-augmented pipelines achieve statistically significant improvements in Mean Time-to-Detect (MTTD) and Mean Time-to-Remediate (MTTR) while maintaining precision $\geq$ 95% for critical vulnerabilities (CVSS $\geq$ 7.0) [21] and reducing false positives by $\geq$ 80% compared to conventional tools.

5.2 Data Collection and Dataset Composition

Source Corpus and Collection Methods :

The study aggregates data across the complete software delivery lifecycle, encompassing 8,447 commits, 2,156 pull requests, 38,920 build executions, 1,247 container images, 892 Infrastructure-as-Code (IaC) artifacts, 12,340 production deployments, 156,890 security events, and 147 confirmed vulnerabilities spanning 12 months . Raw data collection leverages automated integration points, which include Git hooks for version control-related events, API integration with CI/CD platforms (e.g., Jenkins, GitLab CI), API integration with container registries (e.g., Docker, Amazon ECR), Kubernetes audit logs, and Security Information and Event Management (SIEM) solutions (e.g., Splunk, Datadog). Data retention policies preserve raw data for 12 months and archival artifacts for 24 months to support model lineage auditing and reproducibility verification .

Feature Engineering Pipeline Code-Level Features: For each commit, abstract syntax trees (ASTs) are extracted and analyzed to compute control flow complexity according to McCabe's cyclomatic complexity metric [22]:

$$C_{cc} = e - n + 2p \tag{1}$$

where e is the number of edges, n nodes, and p connected components in the control flow graph.

Data dependency chain length is computed as $D_{max} = \max_i(\text{chain}_i)$ across all data flows . Dangerous API calls (SQL operations, command execution, file I/O) are tallied as $A_{\text{danger}} = |\{a \in A : a \in \text{HighRiskAPIs}\}|$. Taint propagation score is computed as $T_{\text{score}} = \frac{\text{propagated_sources}}{\text{total_sources}}$.

Code embeddings are generated using pre-trained CodeBERT models [12], producing 768-dimensional vectors $e_{\text{code}} \in \mathbb{R}^{768}$. Token-level entropy is computed

as $H = -\sum_i p_i \log(p_i)$ where p_i is the probability of token i in the sequence .

Container Image Features: Base image risk is quantified via age ($A_{\text{base}} =$ build_date − release_date) and known vulnerability count $V_{\text{base}} = |\{v \in \text{NVD} : v \in \text{base_image}\}|$ [21]. Update frequency over the preceding 6 months is $U_{\text{freq}} = \frac{\text{updates_count}}{180}$. Package inventory analysis yields total package count P_{total}, outdated packages (¿90 days old) P_{old}, and high-risk packages P_{high} from security blocklists. Vulnerability density per image is $\rho_{\text{vuln}} = \frac{\sum_i \text{CVSS}_i}{P_{\text{total}}}$ [2]. Configuration risk score is computed as:

$$R_{\text{config}} = \sum_j w_j \cdot x_j \tag{2}$$

where $x_j \in \{0, 1\}$ represents binary security configuration flags and w_j are learned weights from training data [23].

Infrastructure-as-Code Risk Metrics: Network security violations are quantified via internet-facing unrestricted ports ($I_{\text{unrestricted}} = |\text{ports} \cap \{0.0.0.0/0\}|$) and missing network segmentation ($S_{\text{missing}} = 1$ if isolated subnets ¡ expected) . IAM policy weaknesses include overprivileged principals ($P_{\text{over}} = |\text{wildcard_actions} \cup$ wildcard_resources|), missing multi-factor authentication enforcement (M_{missing}), and privilege escalation path length
($E_{\text{score}} = \text{shortest_path_to_admin}$).
$Encryptiongapscoreaggregatesunencrypted transitprotocolsandplaintextstorage$:

$E_{\text{gap}} = \frac{\text{unencrypted_protocols+plaintext_columns}}{\text{total_protocols+total_columns}}(3)$

Runtime Telemetry Baseline: Unsupervised anomaly detection establishes per-service operational baselines over months 1–3, computing mean μ_{req} and standard deviation σ_{req} for request rates, error rates, network bytes, and latency percentiles (p50, p95, p99) [10]. Anomaly scores are computed using the Mahalanobis distance [24]:

$$A_{\text{mahal}}(\mathbf{x}) = \sqrt{(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})} \tag{4}$$

Observations exceeding threshold t (optimized via validation data) are flagged as anomalous [13].

5.3 Baseline and Treatment Pipeline Configurations

Baseline (Conventional DevSecOps) The baseline pipeline utilizes industry-standard security tools such as SonarQube for SAST [25], which is configured to fail on every merge request with hard failures for blocker-level issues; Trivy for container scans [26], which is performed during image builds and fails on issues at HIGH+ severities; OWASP ZAP for DAST [27], which runs nightly on the staging environment at the advisory level; Checkov for validation of Infrastructure-as-Code templates; and ELK Stack for implementing rule-based alerting for runtime monitoring. During the observation period from months 7

to 12, the baseline pipeline yielded around 1,847 alerts per day. The documented false-positive rate was found to be around 62%, which needed to be manually triaged and took around 4.2 analyst-hours per day [2, 5].

Treatment (AI-Augmented Pipeline) The treatment configuration deploys four specialized AI components operating in parallel: (1) an XGBoost-based [30] code vulnerability classifier accepting code embeddings and AST features; (2) a Random Forest + SVM ensemble for container risk scoring; (3) an Isolation Forest [31] anomaly detector for runtime telemetry; and (4) a Deep Q-Network (DQN) [32] reinforcement learning agent for automated remediation decisions.

The "Deep Q-Network" (DQN) architecture is based on the implementation of a fully connected neural network with three layers and Rectified Linear Unit (ReLU) activation functions [33]. It takes in a state vector of size 256, which contains code risk scores, container risk scores, anomaly scores, deployment metadata, and historical incident features, and produces Q-values for five distinct actions - `ALLOW`, `WARN`, `BLOCK_BUILD`, `BLOCK_DEPLOYMENT`, and `ROLLBACK`. The reward function is designed as a weighted combination of three components:

$$R_t = w_{\text{sec}} \cdot R_{\text{sec}} + w_{\text{vel}} \cdot R_{\text{vel}} + w_{\text{comp}} \cdot R_{\text{comp}} \tag{5}$$

where $R_{\text{sec}} = -10$ if a vulnerability reaches production, $+5$ if correctly blocked, and 0 otherwise; $R_{\text{vel}} = -1$ for each hour of deployment delay; and $R_{\text{comp}} = +10$ if action maintains compliance, -20 otherwise. Weights are set to $w_{\text{sec}} = 1.0$, $w_{\text{vel}} = 0.1$, and $w_{\text{comp}} = 1.5$ to prioritize security and compliance over deployment velocity. The agent is trained using experience replay with a buffer size of 10,000 transitions, minibatch size of 64, learning rate $\alpha = 0.001$, discount factor $\gamma = 0.99$, and ϵ-greedy exploration with ϵ decaying from 1.0 to 0.01 over the first 50,000 training steps [14, 15].

6 Results and Evaluation

6.1 Detection Performance

The code vulnerability detection model achieved the following test set performance metrics on a held-out dataset of 1,247 code commits:

$$\text{Precision} = \frac{287}{287 + 18} = 0.941 \ (94.1\%) \tag{6}$$

$$\text{Recall} = \frac{287}{287 + 23} = 0.926 \ (92.6\%) \tag{7}$$

$$F1 = 2 \cdot \frac{0.941 \times 0.926}{0.941 + 0.926} = 0.933 \tag{8}$$

For critical vulnerabilities (CVSS ≥ 7.0) [21], the model demonstrated enhanced performance:

$$\text{Recall}_{\text{critical}} = \frac{156}{156 + 5} = 0.969 \ (96.9\%) \tag{9}$$

$$\text{Precision}_{\text{critical}} = \frac{156}{156 + 3} = 0.981 \ (98.1\%) \tag{10}$$

These results show that the model meets the hypothesis requirement of precision 95% for critical vulnerabilities. The container risk scoring ensemble was also shown to have an AUC-ROC of 0.947 on the validation set of 1,247 images, with a false positive rate of 8.2% when optimized to obtain 95% recall. The Isolation Forest anomaly detector, when used on runtime telemetry, was shown to have a precision of 0.883 and a recall of 0.891 on a labeled evaluation set of 4,567 time-series windows, which equates to an 80.3% reduction in false positives compared to the rule-based ELK alerting baseline (IsolationForest 2008). [31].

6.2 Statistical Significance Testing

MTTD Improvement: Paired t-test Null Hypothesis (H_0): $\mu_{\text{baseline_MTTD}} = \mu_{\text{AI_MTTD}}$

Alternative Hypothesis (H_1, one-tailed): $\mu_{\text{baseline_MTTD}} > \mu_{\text{AI_MTTD}}$

Test Results [35]:

- Baseline mean: 1,255 minutes (SD = 289)
- AI mean: 68 minutes (SD = 34)
- Sample size: $n = 187$ vulnerabilities detected

$$t = \frac{1255 - 68}{\sqrt{\frac{289^2}{187} + \frac{34^2}{187}}} = \frac{1187}{20.3} = 58.4 \tag{11}$$

p-value: $p < 0.0001$ (highly significant, ***)

95% Confidence Interval for difference: [1,133, 1,241] minutes

Effect Size (Cohen's d) [36]:

$$d = \frac{1255 - 68}{\sqrt{\frac{(187-1)\cdot 289^2 + (187-1)\cdot 34^2}{187+187-2}}} = 4.24 \text{ (very large)} \tag{12}$$

A very large effect size (Cohen's $d = 4.24$) indicates that the observed improvement is not only statistically significant but also practically meaningful, representing a fundamental shift in detection capabilities.

MTTR Improvement: Mann-Whitney U Test For MTTR, a non-parametric Mann-Whitney U test was applied due to the right-skewed distribution of remediation times [37]:

Null Hypothesis (H_0): Distribution of MTTR is identical between baseline and AI pipelines

Alternative Hypothesis (H_1): AI pipeline has lower MTTR than baseline

Test Results: $U = 2,847$, $n_1 = 187$, $n_2 = 187$, $p < 0.0001$ (highly significant) The median MTTR for baseline was 18.4 hours versus 7.2 hours for AI-augmented, representing a 61.5% reduction.

6.3 Operational Efficiency

Mean Time-to-Detect (MTTD) in minutes [4]:

Table 2. MTTD Comparison: Baseline vs. AI-Augmented

Severity	Baseline	AI-Aug	Δ	% Impr.
Critical	342	18	-324	-94.7%
High	618	32	-586	-94.8%
Medium	1,204	67	-1,137	-94.4%
Low	2,856	156	-2,700	-94.5%
Mean	**1,255**	**68**	**-1,187**	**-94.6%**

Mean Time-to-Remediate (MTTR) in hours [4]:

Table 3. MTTR Comparison: Baseline vs. AI-Augmented

Severity	Baseline	AI-Aug	Δ	% Impr.
Critical	3.2	1.1	-2.1	-65.6%
High	8.1	3.4	-4.7	-58.0%
Medium	24.6	9.2	-15.4	-62.6%
Low	72.0+	28.0	-44.0+	-61.1%+
Mean	**27.0**	**10.4**	**-16.6**	**-61.5%**

6.4 Production Security Outcomes

Six-Month Evaluation (Months 7–12):

Table 4. Production Security Outcomes: Baseline vs. AI-Augmented

Metric	Baseline	AI-Augmented
Exploited vulnerabilities	7	1
Undetected in production	18	2
False positive rate	62%	12.3%
Daily alerts	1,847	487
Manual triage effort	4.2 hrs/day	0.9 hrs/day
Total incident cost	$3.22M	$0.26M
Cost savings	$2.96M (91.9% [34])	
FP reduction	80.2%	
Analyst time saved	78.6%	

The RL agent converged after ∼35,000 training episodes, with action distribution: `ALLOW` (72.3%), `WARN` (18.4%), `BLOCK_BUILD` (6.2%), `BLOCK_DEPLOYMENT` (2.4%), `ROLLBACK` (0.7%). Post-hoc analysis confirmed 94.6% of `BLOCK` and `ROLLBACK` actions were retrospectively validated as correct, indicating high policy quality.

7 Discussion and Implications

The findings suggest that the incorporation of AI in CI/CD processes can result in statistically significant improvements in the rate of vulnerability detection and accuracy, with significant reductions in operational costs. The 94.6% reduction in MTTD and with the 61.5% reduction in MTTR can be related to the current literature on the use of automation in the diagnosis of security incidents, where similar improvements in time to resolution have been reported in the use of AI in the root cause analysis of security incidents (Rajamandrapu, 2025). The calculated effect size for the MTTD improvement, Cohen's d = 4.24, suggests that the improvement is not marginal but transformative.

The 91.9% reduction in security incident costs, or $2.96 million in six months, can be explained by the earlier detection of security incidents, shifting the vulnerabilities further back in the development process, as well as the use of reinforcement learning agents in the remediation of security incidents. The findings support the hypothesis that the use of compliance in the automation of security can effectively support the high deployment frequency of the proposed DevSecOps methodology. The 80.2% reduction in false positive security incidents and the 78.6% reduction in analyst time can be related to the principal challenges of traditional DevSecOps processes (Dunka, 2023).

7.1 Comparative Analysis with Prior Work

Table 5 situates the proposed framework against the most closely related prior works along five dimensions: detection approach, compliance automation, self-healing capability, empirical validation rigor, and reported MTTD/MTTR improvements.

Table 5. Comparison with Related AI-DevSecOps Frameworks

Work	ML Detection	Compliance	Self-Healing	Empirical	MTTD ↓
Alugunuri [1]	✓	–	–	Partial	∼60%
Dunka [2]	✓	–	–	Partial	∼50%
Keller [3]	✓	–	–	Qualitative	N/A
AutoGuard [11]	✓	–	✓	Limited	N/A
Rajamandrapu [42]	✓	–	✓	Full	83.9%
Proposed	✓	✓	✓	**Full**	**94.6%**

The proposed framework uniquely combines compliance-driven gating, multi-modal anomaly detection, RL-based self-healing, and full empirical validation—achieving 94.6% MTTD and 91.9% cost reduction, surpassing Rajamandrapu et al. [42] (83.9% MTTD) whose RCA system lacks proactive CI/CD security and compliance automation. Key limitations include model drift requiring continuous retraining [17], a 12.3% pipeline overhead (47s per build), and reward function sensitivity in the RL agent. Deployment demands GPU resources (∼16 hours/week training) and labeled datasets (∼120 analyst-hours), though at enterprise scale the inference stack autoscales as a Kubernetes sidecar [7], sustaining 38ms median latency across 500+ concurrent builds, with CPU-only distilled variants achieving 91.2% of full-model F1 at 6× reduced cost.

7.2 Incremental Adoption Roadmap

An incremental approach for implementation is recommended rather than the simultaneous implementation of all the components. The proposed incremental adoption plan for the system is as follows:

1. **Phase 1 (Months 1 to 2):** Implement machine learning-based static code analysis and container risk scoring as an advisory (warn only) tool, running alongside existing SAST/DAST tools; gather labeled feedback from security analysts.
2. **Phase 2 (Months 3 to 4):** Enable anomaly detection for runtime telemetry; refine thresholds using feedback from Phase 1, start reinforcement learning agent training using decisions from the shadow pipelines.
3. **Phase 3 (Months 5 to 6):** Activate reinforcement learning-driven automated remediation for low risk actions (`WARN`, `BLOCK_BUILD`); require human approval for high impact actions (`BLOCK_DEPLOYMENT`, `ROLLBACK`).
4. **Phase 4 (Month 7+):** Full automation enabled for all action tiers based on demonstrated agent accuracy, continuous retraining pipeline activated, compliance dashboards integrated with audit workflows.

7.3 Organizational and Cultural Factors

Aside from the technical deployment of AI solutions, organizational alignment is a key aspect that ensures successful adoption. For instance, trust in AI results and findings among AI developers is a crucial aspect. The use of SHAP-based feature importance explanations [18] helps AI engineers to better understand AI results and findings instead of considering AI results and findings as a mandate. The use of a security champion program and a joint working group for security and engineering helps to speed up the feedback loop for AI results and findings and ground truth labeling. This helps to improve the quality of AI models. Organizations that have adopted these cultural changes have shown a higher satisfaction rate for analysts and a lower override rate for AI results and findings in surveys [40].

8 Challenges, Risks, and Limitations

For effective AI models, large amounts of security data with corresponding labels are necessary, which can be difficult to obtain due to issues of privacy, confidentiality, and the low rate of certain types of vulnerabilities [16]. The problem of data imbalance is critical in industrial data sets, where critical vulnerabilities represent merely a small percentage of the total number of identified vulnerabilities, potentially leading to high accuracy in the AI models but missing a large percentage of critical issues in the process. Inaccurate data can result in high false positive and false negative rates, causing the team to distrust the AI models and potentially turning off the automated gates.

Another challenge is the interpretability of the AI models, particularly in regulated industries where the team is required to justify the reason for the blocking of the deployment or the configuration of the AI models [18]. Although the AI models can provide high prediction accuracy, they can lack interpretability, making the validation process difficult for the engineers and auditors [19]. In addition, the deployment of the AI models in the CI/CD process can result in computational costs, which can extend the CI/CD process by several minutes, necessitating the need for critical decisions regarding the deployment of the models in the CI/CD process.

The use of AI models in the CI/CD process also provides more avenues for attackers, where they can use data poisoning, adversarial examples, or denial of service attacks on the expensive AI models [38]. For example, attackers can use data poisoning by injecting logs or codes into the training data of the models, potentially causing the models to underestimate certain types of attacks, or use denial of service attacks by overwhelming the endpoints of the AI models, potentially causing delays in the CI/CD process. Therefore, the integration of the AI models in the CI/CD process requires the implementation of proper governance and access control measures for the data and models used in the process, as well as the need for adversarial testing of the models [39].

9 Future Research Directions

Future work will involve exploring hybrid human-AI collaboration, where AI can perform high-volume pattern recognition, initial triage, and remediation of low- to medium-risk issues, while human analysts can focus on uncertain, high-impact, or strategic decisions [40]. This will require designing interfaces, feedback loops, and escalation paths to allow human analysts to efficiently correct model mistakes and incorporate these corrections into their continuous training loops. Measuring human-AI collaboration effectiveness through collective metrics, such as total detection, analyst time saved, and user-impacting incidents avoided, will also be necessary.

Explainable AI, specifically for security results, represents another promising research direction, which includes methods that can generate human-readable

explanations, highlighting supporting evidence, e.g., particular code paths, configuration settings, or anomalous metric segments [19]. Applying transfer learning, which can allow models trained on vast, diverse codebases to quickly adapt to new projects with limited labeled vulnerabilities, could speed up model time-to-value [**?**]. Further, incorporating threat intelligence feeds, CVEs [41], and adversarial robustness research can allow models to stay up to date with changing threat landscapes while maintaining acceptable model performance, accuracy, and response time for high-frequency CI/CD development cycles [38].

Future research will also include other areas, e.g., federated learning methods to facilitate cooperative model development among teams without sharing source code or vulnerability data, multi-objective optimization to better balance competing objectives in reinforcement learning agents, and causal inference to further improve attribution of security incidents to their causes in complex, distributed systems [42].

10 Conclusion

The integration of artificial intelligence within the CI/CD pipelines can redefine the security function from periodic security checks to a system of continuous operation and adaptation within the context of the proposed DevSecOps framework [3,6]. The orchestration of machine learning-based vulnerability detection, predictive scoring, anomaly detection, and reinforcement learning-based remediation can help organizations improve the rate of vulnerability detection, decrease the mean time to detect (MTTD) and the mean time to remediate (MTTR), and eliminate the need for manual reviews, even for the most rapid deployment scenarios [1,4]. The integration of multi-modal telemetry fusion and causality-based inference, as proposed in the latest research on automated root cause analysis, can improve the overall accuracy and interpretation of security diagnostics.

This research study quantitatively evaluates the efficacy of the proposed AI-based security framework and demonstrates the possibility of achieving 94.6% reduction in MTTD, 61.5% reduction in MTTR, 80.2% reduction in false positives, and 91.9% reduction in security incident costs when compared to the conventional security scenario of the proposed DevSecOps framework. The study found the results to be statistically significant (p 0.0001) with very large effect sizes (Cohen's d = 4.24 for MTTD).

At the same time, the use of such capabilities also presents non-negligible challenges related to the security data, the strength and stability of the models, the transparency of the decisions made by the AI systems, the overhead incurred during the execution of the delivery pipelines, and the governance of the AI-driven activities [16,18]. The empirical research and early industry adoption of such solutions indicate that when the aforementioned challenges are effectively managed, the use of AI-empowered CI/CD pipelines can offer much higher assurance for the security of software systems. The framework proposed in the present work presents a complete blueprint for organizations looking to incorporate AI-driven security automation within the CI/CD pipelines, considering

the conflicting requirements of security assurances, development speed, and regulatory requirements. With the advancement of AI technologies and the changing landscape of cyber attacks, the adaptive approach proposed in the present work presents a viable long-term solution for the security of software delivery pipelines.

References

1. N. Alugunuri: AI for Continuous Security in DevOps (DevSecOps): Integrating Machine Learning into CI/CD Pipelines. *International Journal of Intelligent Systems and Applications in Engineering*, accepted 2024. https://www.ijisae.org/index.php/IJISAE
2. V. K. Dunka: AI-Based Container Security in DevSecOps: Integrating Predictive Analytics for Vulnerability Mitigation in CI/CD Pipelines. *American Journal of Autonomous Systems and Robotics Engineering*, vol. 3, pp. 97–103, Dec. 2023.
3. J. R. Keller: Advancing Automated Security in DevSecOps: Integrating AI, Big Data, and Cloud-Native Approaches for Robust CI/CD Pipelines. *International Journal of Computer Science & Information System*, vol. 10, no. 11, pp. 46–51, Nov. 2025.
4. Integrating AI-Driven Predictive Models and Continuous Security Into DevOps Pipelines: A Unified Framework for Enhanced CI/CD Reliability and Resilience. *International Journal of Networks and Security*, vol. 5, no. 02, pp. 78–85, Nov. 2025.
5. J. S. Al-Farsi: Optimizing CI/CD With AI: Leveraging Machine Learning And DevSecOps For Predictive, Secure, And Efficient Software Delivery. *International Journal of Data Science and Machine Learning*, 2025.
6. K. Gugulotu: Integrating AI/ML into DevSecOps: Strengthening Security and Compliance in Cloud-Native Applications. *International Journal of Computer Engineering and Technology (IJCET)*, vol. 15, issue 5, pp. 1128–1148, 2024.
7. E. S. Munthe: Cloud-Native Transformations: Microservices, Kubernetes, and Security Frameworks in Practice. *Digitus: Journal of Computer Science Applications*, 2025.
8. AI-Driven Cloud Infrastructure: Advances in Kubernetes and Serverless Computing. *International Journal of Advanced Research in Computer Science*, vol. 16, no. 2, 2025.
9. A systematic review on security mechanisms for serverless computing. *Cluster Computing*, vol. 28, article 465, July 2025.
10. Karthick R.: A Comprehensive Survey on AI-Enabled Cloud Security, DevSecOps, and Scalable Digital Infrastructure. *Preprints.org*, 2025.
11. P. Anugula, A. K. Bhardwaj, N. Chhibber, R. Tewari, S. Khemka, P. Ranjan: AutoGuard: A Self-Healing Proactive Security Layer for DevSecOps Pipelines Using Reinforcement Learning. arXiv preprint, Dec. 2025. https://arxiv.org/abs/2501.04240
12. Z. Feng et al.: CodeBERT: A Pre-Trained Model for Programming and Natural Languages. In: Proc. Findings of EMNLP, pp. 1536–1547, 2020. https://doi.org/10.18653/v1/2020.findings-emnlp.139
13. M. Du, F. Li, G. Zheng, V. Srikumar: DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning. In: Proc. ACM CCS, pp. 1285–1298, 2017. https://doi.org/10.1145/3133956.3134015

14. V. Mnih et al.: Human-level control through deep reinforcement learning. *Nature*, vol. 518, pp. 529–533, 2015. https://doi.org/10.1038/nature14236
15. R. S. Sutton, A. G. Barto: *Reinforcement Learning: An Introduction*, 2nd ed. MIT Press, 2018.
16. D. Sculley et al.: Hidden Technical Debt in Machine Learning Systems. In: Proc. NIPS, pp. 2503–2511, 2015. https://proceedings.neurips.cc/paper/2015/hash/86df7dcfd896fcaf2674f757a2463eba-Abstract.html
17. N. Lu, J. Lu, G. Zhang, R. L. de Mantaras: A Concept Drift-Tolerant Case-Base Editing Technique. *Artificial Intelligence*, vol. 230, pp. 108–133, 2016. https://doi.org/10.1016/j.artint.2015.09.009
18. R. Guidotti, A. Monreale, S. Ruggieri, F. Turini, F. Giannotti, D. Pedreschi: A Survey of Methods for Explaining Black Box Models. *ACM Computing Surveys*, vol. 51, no. 5, 2019. https://doi.org/10.1145/3236009
19. M. T. Ribeiro, S. Singh, C. Guestrin: Why Should I Trust You? Explaining the Predictions of Any Classifier. In: Proc. ACM SIGKDD, pp. 1135–1144, 2016. https://doi.org/10.1145/2939672.2939778
20. C. Wohlin et al.: *Experimentation in Software Engineering*. Springer, 2012.
21. Common Vulnerability Scoring System (CVSS) v3.1 Specification. FIRST, 2019. https://www.first.org/cvss/specification-document
22. T. J. McCabe: A Complexity Measure. *IEEE Transactions on Software Engineering*, vol. SE-2, no. 4, pp. 308–320, Dec. 1976.
23. L. Breiman: Random Forests. *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001. https://doi.org/10.1023/A:1010933404324
24. P. C. Mahalanobis: On the generalized distance in statistics. *Proc. National Institute of Sciences of India*, vol. 2, no. 1, pp. 49–55, 1936.
25. SonarQube Documentation. SonarSource SA, 2024.
26. Trivy: A Simple and Comprehensive Vulnerability Scanner for Containers. Aqua Security, 2024.
27. OWASP Zed Attack Proxy (ZAP). OWASP Foundation, 2024.
28. Checkov: Static code analysis tool for Infrastructure-as-Code. Bridgecrew/Palo Alto Networks, 2024.
29. Elastic Stack (ELK). Elastic N.V., 2024.
30. T. Chen, C. Guestrin: XGBoost: A Scalable Tree Boosting System. In: Proc. ACM SIGKDD, pp. 785–794, 2016. https://doi.org/10.1145/2939672.2939785
31. F. T. Liu, K. M. Ting, Z.-H. Zhou: Isolation Forest. In: Proc. IEEE ICDM, pp. 413–422, 2008. https://doi.org/10.1109/ICDM.2008.17
32. V. Mnih et al.: Playing Atari with Deep Reinforcement Learning. arXiv:1312.5602, 2013. https://arxiv.org/abs/1312.5602
33. I. Goodfellow, Y. Bengio, A. Courville: *Deep Learning*. MIT Press, 2016.
34. Cost of a Data Breach Report 2024. IBM Security, 2024. https://www.ibm.com/reports/data-breach
35. J. Cohen: *Statistical Power Analysis for the Behavioral Sciences*, 2nd ed. Lawrence Erlbaum, 1988.
36. J. Cohen: A power primer. *Psychological Bulletin*, vol. 112, no. 1, pp. 155–159, 1992.
37. H. B. Mann, D. R. Whitney: On a test of whether one of two random variables is stochastically larger than the other. *Annals of Mathematical Statistics*, vol. 18, no. 1, pp. 50–60, 1947.
38. N. Papernot et al.: The Limitations of Deep Learning in Adversarial Settings. In: Proc. IEEE EuroS&P, pp. 372–387, 2016. https://doi.org/10.1109/EuroSP.2016.36

39. I. J. Goodfellow, J. Shlens, C. Szegedy: Explaining and Harnessing Adversarial Examples. In: Proc. ICLR, 2015. https://arxiv.org/abs/1412.6572
40. E. Kamar: Directions in Hybrid Intelligence: Complementing AI Systems with Human Intelligence. In: Proc. IJCAI, pp. 4070–4073, 2016. https://www.ijcai.org/Abstract/16/585
41. National Vulnerability Database. NIST, 2024. https://nvd.nist.gov
42. A. K. Rajamandrapu et al.: AI-Automated Root-Cause Analysis for Cloud Incidents. In: Proc. IEEE International Conference on Cloud Computing and AI Systems, 2025.
43. H. Apuri, C. Yepuri: Design of Multi Agent Autonomous Workflow Systems using Agentic AI Frameworks. In: Proc. IEEE International Conference on Advances in Urban Computing (ICAUC), 2026. https://doi.org/10.1109/ICAUC68182.2026.11441280
44. M. M. R. Chinthala, H. Apuri, K. Bitra: Behaviour-Aware Hybrid Deep Networks for Detecting Zero-Day and Ransomware Threats. *International Journal of Innovative Technology and Exploring Engineering (IJITEE)*, vol. 15, no. 5, Apr. 2026. https://doi.org/10.35940/ijitee.E1246.15050426
45. H. Apuri, M. Aurangabadkar, S. Goel, M. M. R. Chinthala, C. Yepuri: Advancing Infrastructure-as-Code Resilience through Generative AI Agents for Predictive Remediation and Autonomous Security Enforcement. *International Journal of Engineering and Advanced Technology (IJEAT)*, vol. 15, no. 4, Mar. 2026. https://doi.org/10.35940/ijeat.D4756.15040326
46. H. Apuri et al.: Self-Healing Infrastructure: Autonomous LLM Agents for Real-Time Remediation of Configuration Drift and Security Misconfigurations in IaC Deployments. *International Journal of Innovative Technology and Exploring Engineering (IJITEE)*, vol. 15, no. 4, Mar. 2026. https://doi.org/10.35940/ijitee.D4757.15040326
47. B. Mohan, V. R. Surasani, R. Kumar: Autonomous Data Stewardship: Multi-Agent AI for Real-Time Master Data Management in Financial Services. In: Proc. 2026 IEEE 16th Annual Computing and Communication Workshop and Conference (CCWC), Jan. 2026. https://doi.org/10.1109/CCWC67433.2026.11393832